Error Detection Capabilities of Cyclic Redundancy Checks

Brian Mearns

2011 Jan 10

Abstract

An attempt to describe mathematically the error detection capabilities of Cyclic Redundancy Checks, also known as CRCs. The author is fairly knowledgeable of the topic, but is not an expert. Basic knowledge of modulo arithmetic and the application of CRCs is assumed.

Content

Given a message M and generator polynomial g, the CRC value C is the remainder when M is divided by g using polynomial arithmetic:

$$\frac{M}{g} = Q + \frac{C}{g} \tag{1}$$

Where Q is the quotient and $C < g$.

Consider applying an error e to the message M to get the erroneous message $\tilde{M} = M + e$. Computing the "erroneous" CRC $\tilde{C}$ of the erroneous message $\tilde{M}$ is done the same way, dividing it by g and taking the remainder:

$$\frac{\tilde{M}}{g} = \tilde{Q} + \frac{\tilde{C}}{g}$$

where $\tilde{C} < g$ and $\tilde{C}$ is the "erroneous" CRC.

Expanding $\tilde{M}$ into our definition based on the orginal message M and the error e, we have:

$$\frac{M + e}{g} = \tilde{Q} + \frac{\tilde{C}}{g}$$

Of course, we can split this quotient into a sum of two quotients:

$$\frac{M+e}{g} = \frac{M}{g} + \frac{e}{g}$$

And now substituting in from equation 1 we get:

$$\frac{M+e}{g} = Q + \frac{C}{g} + \frac{e}{g} = Q + \frac{C+e}{g} \tag{2}$$

Note the remainder of the division $(M+e)/g$ is not necessarily equal to $C+e$, but they are equivalent under modulo g (likewise, the quotient of the division is not necessarily equal to Q). In other words, $C+e$ may actually be larger than g, in which case we could pull additional whole g's out of the fraction to add to Q. In reality the remainder (i.e., the erroneous CRC) is given by:

$$\tilde{C} = (C+e) \bmod g$$

Which can of course be split up:

$$\begin{aligned}\tilde{C} &= (C+e) \bmod g \\ &= ((C \bmod g) + (e \bmod g)) \bmod g \\ \tilde{C} &= (C + (e \bmod g)) \bmod g\end{aligned}$$

Where the last line follows because C is the remainder of division by g and is therefore strictly less than g.

From this, we can see that the erroneous CRC, $\tilde{C}$, is equal to the original CRC, C, if and only if $e \bmod g$ is equal to 0, or in other words, if e is a multiple of the generator polynomial g.

This underlies the error detecting capabilities of CRC: any error e that is not a multiple of the generator polynomial g will cause the remainder of the division (i.e., the CRC value) to change, and will therefore be detected as an error. Errors that are multiples of g will, on the other hand, slip through unnoticed.

Consider what happens if we apply an error e that is strictly smaller than the polynomial: $e < g$. Remember that in our binary polynomial arithmetic, this means that e has strictly fewer significant bits than g. Clearly, a non-zero value less than g cannot be a multiple of g, so any such error will necessarily be detected by the CRC.

However, this only covers errors in the lowest n bits (where the generator g has $n+1$ bits). What happens if we shift this error e up to some other location in the message? Shifting is of course equalivalent to multiplication, but since we're using polynomial division, it is multiplication by a power of the variable x. Specifically, multiplying e by x^i will shift e up by i bits (which is easy to see if you remember that the "bits" of e are actually the coefficients of a polynomial).

Now the actual error that is applied is equal to ex^i, where e is still strictly less than g. The same logic applies though: If we replace e in the expressions above with ex^i, we see that the error will be detected if and only if ex^i is a multiple of g.

In other words, if ex^i divided by g has a non-zero remainder, it will be detected as an error: if it has a remainder of 0, it will not be detected. This quotient can of course be split into the product of two quotients:

$$\frac{ex^i}{g} = \frac{e}{g}\frac{x^i}{g}$$

Now we can expand each of these into a quotient and a remainder, but notice that since we have stipulated that $e < g$, the quotient for that piece will be 0 and the remainder will simply be e:

$$\frac{ex^i}{g} = \left(0 + \frac{e}{g}\right)\left(P + \frac{r}{g}\right)$$

Here, we have expanded each fraction into a quotient and remainder, where P is simply the quotient of x^i divided by g, and r is the remainder of the same. We can pull a factor of $\frac{1}{g}$ out of every term on the right hand side of the equation and get the following:

$$\frac{ex^i}{g} = \frac{e\,(Pg + r)}{g}$$

(because P divided by $\frac{1}{g}$ is of course equal to Pg).

Thus:

$$ex^i = e\,(Pg + r) \tag{3}$$

To summarize where we are, if $e\,(Pg + r)$ is a multiple of g, than so is ex^i (because they are equal by equation 3), and the error ex^i (which is error pattern e, shifted by i bit-places) will therefore be undetected. If it is not a multiple of g, the error will be detected (as shown previously).

Notice that if $e\,(Pg + r)$ is a multiple of g, then taken modulus g it will be equal to 0:

$$\begin{aligned}
e\,(Pg + r) \bmod g &= (e \bmod g)\,((Pg + r) \bmod g) \bmod g && (4)\\
&= (e)\,((Pg \bmod g) + (r \bmod g)) \bmod g && (5)\\
&= e\,(0 + (r \bmod g)) \bmod g && (6)\\
&= e\,(0 + r) \bmod g && (7)\\
&= er \bmod g && (8)
\end{aligned}$$

Equation 4 is a property of products under modulus. Generically:

$$ab \bmod M = ((a \bmod M)\,(b \bmod M)) \bmod M$$

A similar treatment applies to sums under modulus, which is applied to $(Pg + r)$ from equation 4 to equation 5. The other simplification in equation 5 follows because we have stipulated that $e < g$ and therefore $e \bmod g = e$.

Equation 6 is simply observing that the product Pg is a multiple of g and is therefore equal to 0 when taken modulus g.

Equation 7 follows because we have defined r as the remainder when x^i was divided by g, and we therefore have $r < g$. The results are simplified in equation 8.

And so it comes down to this: if $er \bmod g$ is equal to 0, the error ex^i will be undetected, otherwise, it will be detected. In otherwords, if and only if the product er is a multiple of g, the error will be undetected.

First, let us look at the trivial case where er is equal to 0 (i.e., $0g$). This requires that e or r be equal to 0. If e is equal to 0, it means there was no error to detect, so we can ignore that.

On the other hand, r will be equal to zero if and only if x^i is divisible by g (because r is the remainder of this division). Note that generically x^i is only evenly divisible by x^k, for any $k \leq i$, so as long as g is not equal to x^k for any $k \leq i$, x^i will not be divisible by g, and so r will not be zero. In otherwords, g should have more than one non-zero coefficient (so that it takes the form $x^k + p(x)$ for some non-zero polynomial $p(x)$), and that is sufficient for ensuring that r will not be 0.

Now consider the non-trivial cases, where $er = kg$ for some integer $k > 0$. Generally, there is no reason to assume this couldn't happen. However, consider what happens if g is prime. If g is prime, then the only way to expand kg is by expanding k into its factors (because g is prime, it cannot be expanded into factors). For instance, if $k = mn$, then we have $er = kg = mng$, which looks like $er = m(ng)$. In otherwords, no matter how you break this up, one of the terms in the product (i.e., one of e and r) must be a non-zero multiple of g. But since we have already seen that both e and r are strictly less than g, this is not possible. In otherwords, er cannot possibly be equal to kg if g is prime, which means (if you trace the logic back far enough) that ex^i cannot be a multiple of g and therefore cannot slip through undetected.

In summary, if the generator polynomial g is prime and has at least two non-zero coefficients, then any error e which is strictly smaller than g will be detected by the CRC algorithm.

In practical terms, this means that an n-bit CRC (where the CRC value is n bits long, and the generator polynomial is $n + 1$ bits long) will be able to detect any single error vector that has no more than n signigicant bits, no matter where in the message it is applied, and no matter how long the message is.

Notice that this does not mean it can always detect n bit errors. If those errors are spaced apart in the message, then the error vector e will actually have more than n significant bits, and the above no longer holds. It is still possible for the CRC to detect the error (anytime the error is not a multiple of g), but it is not guaranteed for any particular size of error greater than n.